UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/706,546	11/12/2003	Feng-Wei Chen Russell	RSW920030186US1	6883

23550 7590 01/18/2007

HOFFMAN WARNICK & D'ALESSANDRO, LLC
75 STATE STREET
14TH FLOOR
ALBANY, NY 12207

EXAMINER	
TIMBLIN, ROBERT M	

ART UNIT	PAPER NUMBER
2167	

MAIL DATE	DELIVERY MODE
01/18/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

Advisory Action Before the Filing of an Appeal Brief	Application No. 10/706,546	Applicant(s) RUSSELL ET AL.	
	Examiner Robert M. Timblin	Art Unit 2167	

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

THE REPLY FILED 18 December 2006 FAILS TO PLACE THIS APPLICATION IN CONDITION FOR ALLOWANCE.

1. ☒ The reply was filed after a final rejection, but prior to or on the same day as filing a Notice of Appeal. To avoid abandonment of this application, applicant must timely file one of the following replies: (1) an amendment, affidavit, or other evidence, which places the application in condition for allowance; (2) a Notice of Appeal (with appeal fee) in compliance with 37 CFR 41.31; or (3) a Request for Continued Examination (RCE) in compliance with 37 CFR 1.114. The reply must be filed within one of the following time periods:
   a) ☐ The period for reply expires _____months from the mailing date of the final rejection.
   b) ☒ The period for reply expires on: (1) the mailing date of this Advisory Action, or (2) the date set forth in the final rejection, whichever is later. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of the final rejection.

   Examiner Note: If box 1 is checked, check either box (a) or (b). ONLY CHECK BOX (b) WHEN THE FIRST REPLY WAS FILED WITHIN TWO MONTHS OF THE FINAL REJECTION. See MPEP 706.07(f).

Extensions of time may be obtained under 37 CFR 1.136(a). The date on which the petition under 37 CFR 1.136(a) and the appropriate extension fee have been filed is the date for purposes of determining the period of extension and the corresponding amount of the fee. The appropriate extension fee under 37 CFR 1.17(a) is calculated from: (1) the expiration date of the shortened statutory period for reply originally set in the final Office action; or (2) as set forth in (b) above, if checked. Any reply received by the Office later than three months after the mailing date of the final rejection, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

NOTICE OF APPEAL

2. ☐ The Notice of Appeal was filed on _____. A brief in compliance with 37 CFR 41.37 must be filed within two months of the date of filing the Notice of Appeal (37 CFR 41.37(a)), or any extension thereof (37 CFR 41.37(e)), to avoid dismissal of the appeal. Since a Notice of Appeal has been filed, any reply must be filed within the time period set forth in 37 CFR 41.37(a).

AMENDMENTS

3. ☐ The proposed amendment(s) filed after a final rejection, but prior to the date of filing a brief, will not be entered because
   (a) ☐ They raise new issues that would require further consideration and/or search (see NOTE below);
   (b) ☐ They raise the issue of new matter (see NOTE below);
   (c) ☐ They are not deemed to place the application in better form for appeal by materially reducing or simplifying the issues for appeal; and/or
   (d) ☐ They present additional claims without canceling a corresponding number of finally rejected claims.

   NOTE: _____. (See 37 CFR 1.116 and 41.33(a)).
4. ☐ The amendments are not in compliance with 37 CFR 1.121. See attached Notice of Non-Compliant Amendment (PTOL-324).
5. ☐ Applicant's reply has overcome the following rejection(s): _____.
6. ☐ Newly proposed or amended claim(s) _____ would be allowable if submitted in a separate, timely filed amendment canceling the non-allowable claim(s).
7. ☒ For purposes of appeal, the proposed amendment(s): a) ☐ will not be entered, or b) ☒ will be entered and an explanation of how the new or amended claims would be rejected is provided below or appended.
   The status of the claim(s) is (or will be) as follows:
   Claim(s) allowed: _____.
   Claim(s) objected to: _____.
   Claim(s) rejected: 1-31.
   Claim(s) withdrawn from consideration: _____.

AFFIDAVIT OR OTHER EVIDENCE

8. ☐ The affidavit or other evidence filed after a final action, but before or on the date of filing a Notice of Appeal will not be entered because applicant failed to provide a showing of good and sufficient reasons why the affidavit or other evidence is necessary and was not earlier presented. See 37 CFR 1.116(e).
9. ☐ The affidavit or other evidence filed after the date of filing a Notice of Appeal, but prior to the date of filing a brief, will not be entered because the affidavit or other evidence failed to overcome all rejections under appeal and/or appellant fails to provide a showing a good and sufficient reasons why it is necessary and was not earlier presented. See 37 CFR 41.33(d)(1).
10. ☐ The affidavit or other evidence is entered. An explanation of the status of the claims after entry is below or attached.

REQUEST FOR RECONSIDERATION/OTHER

11. ☒ The request for reconsideration has been considered but does NOT place the application in condition for allowance because: See attached Continuation Sheet.
12. ☐ Note the attached Information Disclosure Statement(s). (PTO/SB/08) Paper No(s). _____
13. ☐ Other: _____.

ALFORD KINDRED
PRIMARY EXAMINER

The request for consideration has failed to place the application in condition for allowance for the following reasons:

The Applicant argues on page 11 of response and in respect to claims 1, 9 , 16, and 24 that Gorelik fails to teach a mining model. The Examiner respectfully disagrees because Gorelik's method of discovering semantics, relationships and mappings between data (paragraphs 0026 and 0030) indicates data mining processes. Furthermore, Gorelik teaches a schema is converted into a normalized relational model in a process used for data analysis to discover relationships between data objects (paragraph 0031). Thus, in Gorelik's method of data mining, the schema in paragraph 0031 equates to the claimed "mining model schema" because it is used in a data mining application (i.e. acqired using a data mining application.

The Applicant further argues on page 12 and respect to claims 1, 9, 16, and 24 that Gorelik fails to teach matching by performing a number of matching processes in a sequence untili a match is found. The Examiner repsectfully disagrees because Gorelik teaches finding a best match in paragraph 0055. Drawing reference 606 mentions finding a best match for each target location. To further explain, in paragraph 0387 and paragraphs 0445-0467, there is disclosed a list of locations for finding a match. In finding a match for each location, it is apparent that a number of matching processes in sequence is performed because a list is used (indicating sequence) to find the best match.

At the Applicant's request, a copy of the Gorelik provisional is included with this response.

6764P001Z *PATENT*

UNITED STATES PROVISIONAL PATENT APPLICATION

for

A METHOD AND APPARATUS FOR SEMANTIC DISCOVERY AND MAPPING BETWEEN DATA SOURCES

Inventor:
Alex Gorelik

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Los Angeles, CA 90026-1026
(408) 720-8598

Express Mail Label No.
EV336582297US

A METHOD AND APPARATUS FOR SEMANTIC DISCOVERY AND MAPPING BETWEEN DATA SOURCES

FIELD OF THE INVENTION

The present invention relates to a method and apparatus for automating the way computer systems, applications, files and databases are integrated. Specifically, the present invention relates to the discovery of semantics, relationships and mappings between data in different software applications, databases, files, or systems.

DESCRIPTION OF RELATED ART

The Information Technology (IT) professionals have performed data and application integration for many years. A typical integration project has three distinct phases: discovery, integration, and maintenance. Discovery phase involves identifying relationships between the systems that need to be integrated. Integration phase involves creating programs or specifications to perform the physical data movement or interfacing. The maintenance phase involves updating and changing the integration programs to correspond to changes in the systems being integrated or to accommodate new integration requirements.

Several prior art patents describe various conventional ways of integrating data across systems.

US05675785 – Hall, et al. 10/7/97 395/613, "DATA WAREHOUSE WHICH IS ACCESSED BY A USER USING A SCHEMA OF VERTICAL TABLES"

This patent describes a system where a layer of logical tables is created and mapped to the physical tables in a data warehouse such that the user specifies queries

against the logical tables to access data in the physical tables. It does not address the problem of discovering relationships and mappings between data in different data sources

US05806066 – Golshani et al, 9/8/98 707/100, "METHOD OF INTEGRATING SCHEMAS OF DISTRIBUTED HETEROGENEOUS DATABASES"

This patent describes a graphical system that allows the user to graphically merge multiple distributed schemas into a global schema. It does not address the use of data to determine the relationship between the two schemas.

US05809297 Kroenke, et al 9/15/98 395/613, "SEMANTIC OBJECT MODELING SYSTEM FOR CREATING RELATIONAL DATABASE SCHEMAS"

This patent describes how to create a relational schema from a semantic object definition.

US06026392 Kouchi et al. 2/15/2000 707/200, "DATA RETRIEVAL METHOD AND APPARATUS WITH MULTIPLE SOURCE CAPABILITY"

This patent describes a system that moves data from source database to target database with a different structure. However, the '392 patent does not describe creating a new data source based on the structure of an existing data source, or moving data from the existing data source to the new one, or creating reports from the new data source.

US06226649 Bodamer, et al 5/1/2001 707/104, "APPARATUS AND METHOD FOR TRANSPARENT ACCESS OF FOREIGN DATABASES IN A HETEROGENEOUS DATABASE SYSTEM"

This patent addresses remote access of data from a heterogeneous database. The '649 patent does not address the determination of how that remote data is related to the data in the local database.

US06339775 Zamanian et al. 1/15/2002 707/101, "APPARATUS AND METHOD FOR PERFORMING DATA TRANSFORMATIONS IN DATA WAREHOUSING"

This patent describes a system and apparatus that extracts, transforms and loads data from one or more data sources to a data warehouse.

US06393424 * Hallman et al, 5/21/02 707/10, "METHOD AND APPARATUS FOR USING A STATIC METADATA OBJECT TO REDUCE DATABASE ACCESS"

This patent describes a system that retrieves metadata from memory and uses it to retrieve data from the database. This patent applies to a single database access, not a relationship between data in different databases or different data tables in the same database.

US22178170A1 Britton, et al 11/28/2002 707/100, "METHOD AND APPARATUS FOR ENTERPRISE APPLICATION INTEGRATION"

This patent addresses heterogeneous data access, not the relationship between heterogeneous data.

WO 01/75679 A1 Scanlon et al. 10/11/2001 G06F 17/30, "A SYSTEM AND METHOD FOR ACCESSING DATA IN DISPARATE INFORMATION SOURCES"

WO 02/073468 A1 - Kil, et al. 9/19/02 G06F 17/30, "AUTOMATIC DATA EXPLORER THAT DETERMINES RELATIONSHIPS BETWEEN ORIGINAL AND DERIVED FIELDS"

The integration phase has been the focus for computer scientists and software vendors for many years. The discovery phase, however, has not been automated and frequently involves a time-consuming manual and cross-functional effort.

SUMMARY OF THE INVENTION

This invention is in the area of data analysis and concerns automatic determination of causal relationships and correlations between various data fields in order to improve the performance of a data analysis process.

Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which

Figure 1 illustrates a typical computer system on which the present invention may be implemented.

DETAILED DESCRIPTION

An apparatus and method is described for the discovery of semantics, relationships and mappings between data in different software applications, databases, files, or systems. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be evident however to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well known structures and devices are shown in blocked diagram form in order to avoid unnecessarily obscuring the present invention.

Reference throughout this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrases "in one embodiment" or "in an embodiment" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

In one embodiment, steps according to the present invention are embodied in machine-executable software instructions, and the present invention is carried out a processing system by a processor executing the instructions, as will be described in greater detail below. In other embodiments, hardwired circuitry may be used in place of, or in combination with, software instructions to implement the present invention.

Overview

The present invention automates the discovery of semantics, relationships and mappings between data in disparate software applications, databases, files or systems. The metadata and the data in the systems are analyzed using a set of rules, techniques and statistics to deduce relationships between the systems.

The schema in the disparate system is converted into a normalized relational model. Normalized schemas are converted into data objects – objects with a unique key and scalar and non-scalar attributes. Metadata and data analysis is used to discover a binding condition that relates the data objects instances across systems. Binding condition is a Boolean expression on attributes of the data objects from one or more source systems and a target system that identifies which instances of the source data objects map to which instances of the target data object. The object instances thus related through the binding condition are said to be bound.

Bound object instances are then analyzed to establish attribute correlation. Finally, transformation discovery is applied to correlated attributes (columns) to discover transformation functions. Conversely, attributes are used to establish binding in the first place.

As will be described in more detail below, the present invention includes three main processes that contribute to the automated discovery and mapping system. The first process discovers the binding condition between tables in disparate systems. The second process uses an established binding condition to discover correlations between the columns of the bound tables. The third process uses an established binding condition and

correlation to discover transformation functions between correlated columns of bound tables.

For simplicity of description, all processes assume one source table and one target table. However, the processes can be applied to multiple source tables by treating such tables as a single view that represents a join of such source tables.

Also for simplicity of description, all processes assume that all source and target tables can be copied in their entirety to a single relational database. More practical implementations involve the use of intelligent sampling that only uses related portions of the source and target tables. Alternatively, federated databases, distributed queries or data Extraction, Transformation and Loading (ETL) tools may also be used to join tables across databases.

Assumptions and Observations

All data in a conventional business enterprise is related one way or another – it is mostly a matter of discovering the relationship.

It is possible to use data to identify relationships between data tables.

Except for several well-understood design patterns (e.g., timestamps, counters, generated keys), accidental correlation between different data sources is highly unlikely in large data sets.

Key Concepts

Data Objects – all the key business entities in the enterprise can be represented as a set of data objects. Relationship between these data objects captures the business rules and workings of an enterprise.

Every piece of data in the enterprise belongs to one or more data objects.

The data objects are present in every system of the enterprise. They can be persisted in the databases or files, exchanged as messages and manipulated by the applications.

Since all enterprises require history and persistence, all the data objects are persisted (stored) somewhere in the enterprise.

Because the data objects represent the enterprise, all the data sources in the enterprise are related to each other through the data objects.

Furthermore, all the systems in the enterprise have to work together irrespective of whether they are integrated through automated means (machine to machine) or through intermediate human processes or both. We know they do because orders get accepted, goods get manufactured, orders get filled, services provided, financial information reported and taxes paid.

While there may be gaps in the data chains because of purely manual processes and record keeping – it is uncommon for any sizable enterprise that any major system is not automated – links and business processes between systems can be manual, but the systems themselves are normally computerized.

Data objects can be identified and normalized through key discovery, primary-foreign key dependency discovery and join key discovery.

Data objects are represented as a set of parent-child tables where the child has a foreign key from the parent. Data Object is an acyclical graph (a tree).

Data object has one and only one root table

Data object can be represented as a left-outer join of every parent and every child without any loss of data

Source Data Objects (SDO) – data objects discovered in a single data source.

Universal Data Objects or Global Data Objects (UDO) – data objects that are persisted in multiple data sources. UDOs are mapped to one or more SDOs.

Binding – is a way to identify the same instance of a UDO across one or more SDOs. For example, if we have a Customer Universal Data Object and a set of Customer Source Data Objects mapped to the Customer UDO, there is a binding condition for each SDO that identifies the same instance of the data object (i.e., the same physical customer) between that SDO and the Customer UDO.

UDO to SDO binding - Binding is always discovered or specified between UDO and an SDO.

SDO to SDO binding - Using transitive closure, we can generate a binding condition between two or more SDOs using the binding conditions between each SDO and UDO. Sometimes, there may not be a direct binding condition between two SDOs and additional SDOs linking the original two need to be included. For example, there may be a lookup table in Source1 that links together customer ids in Source2 with customer ids in Source3. A binding condition between Source2 and Source3 will have to include Source1 and the lookup table.

Binding is essentially a join condition – a federated, full outer join condition - a full outer join between tables in two or more data sources.

We assume that majority of the binding conditions are equijoins between table keys.

Correlation – is a way to discover binding conditions.

Correlation is a way to identify data attributes that represent the same values a different way. For example, one system may represent sales regions as North, South, East and West and another as 1, 2, 3 and 4. However, for the same instance of a data object

such as customer, the region in one system would correspond to a region in the other system – e.g., North would correspond to 1, South to 2, etc.

Most of the time, we do not have to know what the transformation function is to detect correlation.

Given a scalar, stateless transformation function f that can transform a value V1 in System1 into a value V2 in System2 (V2 = f(V1)), for the same instance of a data object in System 1 and System 2, if the same value V1i in system 1 always corresponds the same value of V2i in System 2, there is a correlation between the two.

There are transformation functions that do not produce correlations detectable by value correspondence. These include Aggregate functions.

There are false correlations. These are usually based on generated keys such as integer counters, timestamps, etc. However, these are typically our of sync and do not detect any other correlations and are therefore not picked as binding conditions.

Because we use statistics and attempt to find the best rather than absolute match, we can work with dirty or incomplete information and do not have to identify all possible binding conditions.

Correlations can be between a single column in one data object and one or more columns in another data object. Because transformation functions are scalar, the target has to be a single column.

We assume that a binding condition will have the highest number of attribute correlations between an SDO and a UDO. Thus, we can look at potential binding conditions and see which one gives us the highest number of correlations.

Value Matching – a matching of values in two attributes or columns – returns statistics about how many values in one column have matches in another column.

Bound Matching, Transformation Function Discovery – using a binding condition and correlation between a set of source columns and target columns, try to identify the transformation function that converts source column values into the target column value.

Approach

Given a data source and a set of universal data objects UDOs, map the source to UDOs.

Identify local data objects at the source (SDOs)

Use key analysis to identify keys

Use foreign key and join key discovery to identify relationship between tables

Build trees of tables

Remove cycles

Convert each tree into a source data object SDO

Determine binding conditions between UDOs and SDOs

Perform Value Matching

Determine correlations between data sources through data analysis and statistics

Identify binding condition with the highest correlation

Match instances of data objects across data sources using bound matching

Apply binding condition between a UDO and SDO

Look at correlated columns

Apply transformation functions to see if can get target value from source values

Simple Automatic Mapping Overview

The following simplest method, although not practical for large-scale deployments, captures the essence of the approach and illustrates the techniques that will be used in more complex approaches.

Given:

- Two data sources: Source S and Target T with tables: Ts1...Tsi and Tt1...Ttj respectively, where each table has columns Ct1..Ctn where t is the table number (e.g., CS11..CS1n)
- A set of Value Match Rules (VMR) that can take 2 input tables and determine how many rows in one table match the rows in the other table. The rules can be as simple as equality (that determines if the values in two columns are identical) or very complex. Every VMR returns the following statistics:
    - o Number of rows in Target table that had matches,
    - o Number of distinct values in Target table that had one or more matches.
    - o Number of distinct values in Target tables that had 1 or more mismatches.

Discover mappings between the columns, tables and data sources as follows:

1. Determine value match scores
2. Discover binding conditions
    a. Determine correlation scores – does the same value of S correspond to the same value of T.
3. Discover attribute matches
    a. Determine correlation scores

4. Discover filters – horizontal partitions of the table where a given binding condition or mapping is strong

Simple Process

In the following description we will refer to UDO as a target and SDO as a source.

First, identify potential binding condition columns:

For each target attribute and a set of source attributes, determine Value Match Scores by applying every VMR to every set of source columns – create a list of statistics for each VMR. More precisely,

For each target attribute TAi

For each set of source attributes (SA1,..,SAn)

For every Value Mapping Rule VMRx

Apply VMRx(SA1,..SAn)

If the matching row count > Row Number Match Threshold (e.g., 20%),

Store scores in a Value Match Table (VMT)

The columns with entries in VMT are the potential binding condition predicates.

Validate Binding Conditions using correlations:

Make a list of potential binding conditions by creating every permutation of the binding condition predicates in VMT.

For every potential binding condition BCi,

Construct a binding condition expression BCEi

For every binding predicate BPj with target attribute TAj, Value Match Rule VMRj and source attributes SAj1..SAjn, create a predicate expression of the form TAj = VMRj(SAj1,..,SAjN)

Create a conjunction of all the predicates: TA1 = VMR1(SA11,..,SA1N) and TA2 = VMR2(SA21,..,SA2M) and ...

Create a view of the rows that match the binding condition expression.

I.e., SourceBoundView = select * from Source where exists (select * from Target where BCi) and TargetBoundView=select * from Target where exists (select * from Source where BCi)

Optionally, remove all columns with no data (e.g., all NULL) or with low and high selectivity (selectivity = #unique values/#rows – e.g., a unique key will have a selectivity of 1, a gender column – 2/#rows). Filter out the columns with selectivity < 0.1 or selectivity > 0.9 – they will not give meaningful correlation

Create a list of all possible combinations of SourceBoundView columns that are not part of the binding condition (Csi+1..CSn): {Csi+1}, {Csi+1, Csi+2}, ..., {Csi+1, .., CSn})

Create a list of all possible combinations of TargetBoundView columns that are not part of the binding condition

Determine correlation score between every combination of source and target columns:

Correlation score indicates whether the same value of the source combination corresponds to the same value of the target combination. Note that the values do not have to be the same, as long as the same

value of source columns (e.g., A) always corresponds to the same value of target columns (e.g., B). Correlation Score = % rows where correlation holds * selectivity

Determining the correlation can be done as follows – not efficient, but functionally correct:

For each source column, for each distinct value (CurrentSourceValue)

Create a view ValView as select * from Target where exists (select * from source where BindingCondition and SourceColumn = CurrentSourceValue)

For each target column

Select count (TargetColumn) from ValView group by TargetColumn

Choose the group with highest count, add it to the CorrelationRowCount

At this point CorrelationRowCount contains the count of rows that are correlated. % rows where correlation holds = CorrelationRowCount/Target Row Count

Add all correlation scores for each binding condition

If the highest correlation score < CorrelationThreshold (e.g., 0.6), there is no binding between the tables. Otherwise, choose binding condition with highest correlation score.

Discover Transformation Functions

For each pair of columns with correlation score > CorrelationThreshold, apply Transformatino Rules (TR) to determine if we can deduce a transformation function.

We save all TRs with value match score > TransformatinMatchThreshold (e.g., 80%) as alternate mappings and the TR with highest value match score as the primary mapping

Discover filters – sometimes we can get very accurate mappings if we restrict the data set. We discover potential filters as follows:

Given a binding condition and a TR, go through every source column and identify all columns where if a column C has a set of unique values V1..Vn, so of the values V1..Vi only occur in rows that match the binding condition and the mapping and other values Vi+1..Vn only occur in rows that do not match the binding condition.

We may wish to relax the matching requirement, such that percentage of filter mismatches < FilterThreshold (e.g., 90%) to account for dirty data

Generate filter condition from the value list as follows:

If maximum value among the matching values VmaxMatch = max(V1,..., Vi) is less than the minimum value among the non-matching values VminMismatch = min(Vi+1, Vn), generate the filter of the form C < VminMismatch

Else if VminMatch > VmaxMismatch – generate filter of the form C > VmaxMismatch

Else if there is a small number of values that match the condition, generate a filter condition of the form C in (V1, ..., Vn)

Else, if there is a small number of values that do not match the condition, generate a filter condition of the form C not in (Vi+1, ..., Vn)

If we discover a filter for a table, create a view on the source table as "select * from Source where not(FilterCondition)" and process as a separate table.

5

Complete Mapping Discovery

Complete mapping discovery is more elaborate than the simple process in order to improve the performance. Instead of performing a match between every pair of attributes, we identify the attributes with the highest possibility of matching and then apply the match rules.

Join Condition Discovery – discover which tables are related and how they are related.

Identify all keys in each table – this will help identify natural keys as well as generated primary keys

Identify inclusion dependencies – if table C has a foreign key (columns F1..Fn) from P and any Fi is part of any key of C discovered in previous step, there is an inclusion dependency between C and P where P is the parent and C is the child

Generate Source Data Objects using join conditions.

Build an SDO tree – a tree of tables joined by a parent-child relationships discovered in step 1

Build an SDO view as a full (left?) outer-join of all parents and children in the tree

Perform Schema Match between source and target data objects. Identify attributes with high potential of matching.

Discover Binding conditions.

Identify tables that have potential matches

Perform correlation analysis – identify groups of columns that have a high degree of correlation

Perform Transformation discovery on attributes with high degree of correlation

Perform Filter Condition Discovery on attributes with high degree of correlation (as described in the simple method).

Join Condition Discovery

Join Condition Discovery (JCD) discovers which tables are related and how they are related. First step in JCD is to identify all the keys in each table. The keys may be declared as primary keys, may be defined as unique indices or may have to be discovered by examining the data for column sets that would constitute a unique index.

As an example of one embodiment, we will use the Northwind sample database that comes with MSAccess and MS SQL Server to illustrate the processes.

Key Discovery

We discover unique keys as follows:

If a table has a primary key consisting of columns P1, ..., Pn, add it to the unique key set K, K = {(P1, ..., Pn)}

Northwind.Suppliers.SupplierId

If there is a unique index for a table on columns I1,..., Im, add the columns of the index to K as a key: K = {(P1, ..., Pn), (I1, ..., Im)}

Check cardinality of every column. If the relation has R rows and a column Cn has R unique values, then it is unique – add it to K: K = {(P1, ..., Pn), (I1, ..., Im), (Cn)}

Northwind.Suppliers.CompanyName, ContactName, Address, PostalCode, Telephone

Check cardinality of all column (C1,...,CnonInKCount) not in K already

Determine cardinality = select count distinct(C1 | ... | CnotInKCount) from T

If cardinality (the count of distinct values of all column not in K) < R, there are no more unique keys in T

If it is R, determine all minimal subsets that are unique as follows:

Given a set of columns C1,...,Cn with cardinality R, check cardinality of subsets (C1, ..., Cn-1); (C1, C3, ..., Cn), (C1, C2, C4, ..., Cn) and so on.

If the cardinality of all subsets is less than R, then C1,...,Cn is the minimal subset – add it to K.

Otherwise, repeat this exercise for each subset with cardinality of R

Cardinality of Northwind.Suppliers is 29. Cardinality(Suppliers.ContactTitle, Region, Country, Fax, HomePage) = 29, but selectivity of any subset of the columns < 29. Therefore, there is a unique key consisting of ContactTitle, Region, Country, Fax, HomePage.

Foreign Key Discovery

We discover foreign keys through data source metadata and through data-driven discovery. Discovery is performed as follows:

Perform value match between every key column discovered in Key Discovery Step of every table.

If the hit percentage (#rows that match/#rows in a table) is > ForeignKeyThreshold (e.g., 50%) for either table, then we say there is a *potential* foreign key dependency between the two systems on this column

Match full key from one table to full or partial key in the other table. We do not want to match partial keys to partial key (e.g., Zip code in customers will have a

high match rate with Zip code in suppliers – neither is a key in itself and the match is meaningless unless a full key from either relation can be matched).

For each key in Table1, if every column in the key has a corresponding potential foreign key in Table 2, perform correlation analysis to make sure that the columns in the same row in Table1, match the columns in the same row in Table 2. If the correlation score is lower than 80% of the smallest match score for individual columns, assume this is a false foreign key

After correlation analysis, we may still be left with multiple keys. Perform correlation analysis across different keys. If there is a match (> 80% of minimum correlation), we combine the keys into a single foreign key. Otherwise, we leave them as separate potential foreign keys

Determine which table is primary and which is foreign

If there is a primary-foreign key relationship between a full key of P, but only a partial key of C, then P is the parent and C is the child and the primary-foreign key relationship between them is said to be identifying

If the match columns include a full key of P and a full key of C, then P and C are peers and the relationship is said to be a partition relationship.

Note that because we may discover multiple join keys, the same two tables may have multiple instances of each type of relationships: Table 1 as parent, Table 2 as child; Table 2 as parent and Table 1 as child; and Table 1 and Table 2 as peers. Each relationship will have a different join key.

Inclusion Dependency Discovery

Identify inclusion dependencies – if table C has a foreign key (columns F1..Fn) from P and any Fi is part of any key of C discovered in previous step, there is an inclusion dependency between C and P where P is the parent and C is the child

False Join Condition Discovery

Key discovery helps identify unique keys for each table. Inclusion discovery identifies foreign keys that are parts of a key for a table. There are many false foreign keys (e.g., custNum may go from 1..N and productNum may go from 1..M. A table T may have a column Num that goes from 1..L where L <N and L<M). We can try to resolve ambiguity using metadata matching. However, in the worst case, we will create an inclusion dependency between Cust and T on CustNum=Num and between Product and T on ProductNum=Num. These false dependencies are unlikely to yield any correlations and are mostly performance issues.

Source Data Object Construction

Once we have all the parent-child relationships identified, we construct source data objects as follows:

We have a list of tables in the data source. For each table, we have the following relationships:

Parent List – a list of tables that have an identifying foreign key in this table

Peer List – a list of tables that have a partition relationship with this table

Child List – a list of tables that have an identifying foreign key that corresponds to a primary key in this table

Join the peers

Create a view by doing a full outer join of all peer tables on their shared primary key

The view's parent and child lists are concatenations of the peer's parent and child lists

Remove the peers from the table list, add the combined view to the table list

Remove cycles

Mark all tables that do not have any children or any parents as "Clear". Mark all other tables as "Potential Cycles"

Order the tables with Potential Cycles by the number of parents. Start with the tables with the least number of parents.

For each table with Potential Cycles starting table S follow the child links and resolve the cycles using the Cycle Resolution Process.

Create Source Data Objects (SDOs)

Make each table that does not have any identifying foreign keys a source data object – it will be the "root" of the SDO

For each SDO, add all the descendants of the root (children, children's children, etc.)

If there are any cycles, break the cycles by creating two separate views on a table causing a cycle

Each SDO can be "unfolded" into a single table by performing an outer join between every parent and its children. Then we can perform matching on the resulting flat table.

Alternatively, we can further subdivide tables into second or even third normal forms and perform matching on smaller relations.

Cycle Resolution Process

Given a starting table S, a current table T and a list of processed tables P, for each table C in T's child list:

If C is marked as Clear – stop

If C is the original starting table (S) – we have a cycle, resolve it by creating another instance of S – a table S'. Replace S with S' in T's child list. Remove T from the parent list of S and add it to S' parent list.

If C is already in P stop – we will not address this cycle here, we will address it when we process C or T

Otherwise, add C to P, call Cycle Resolution Process with T=C

At the end of the process, mark S as Clean

Manually Adjusting Source Data Objects

A user can create new data objects or add tables to existing source data objects using a graphical user interface.

In order to create a source data object, the user needs to specify:

- Data Object Name
- Root Table

In order to add a table to an existing Source Data Object, the user needs to specify:

- Table being added
  - The parent for the table
  - Join Condition to join the table with its parent

Universal Data Object Construction

Universal data objects span one or more data sources. Universal Data Objects are created using the Mapping Studio or by auto-mapping them to the Source Data Objects. Universal Data Object is a table tree similar to the Source Data Object, except it includes mappings for each table and each attribute to any number of Source Data Objects for any number of Data Sources.

Universal Data Object Definition

Universal Data Object is a common data object representing different Source Data Objects in a single integrated structure. Its data model is a tree of tables related to each other through identifying reference relationship. Data object has the following properties:

- Table Tree where each parent table has zero or more child tables. There is a unique join condition to join each parent and child table pair – typically an identifying primary-foreign key condition
- Mapped Source Data Objects – the local data objects from different data sources that have been mapped to this data object. For each, we will have
    - Mapping for each table in the SDO including:
        - Binding condition
        - Attribute mappings

In addition, the following terms are used to describe a data object:

- Root table is the root of the Table Tree. Root table does not have any parents

- Nested Table – is any table in the Table Tree that is not a root table.
- Leaf Table – a table in the Table Tree that does not have any children
- Primary key - the primary key of the root table
- Extended table – a full left outer join of all the tables in a table tree achieved by starting with the root table and performing a left outer-join on all the child tables and then their child tables, etc. such that the parent is always the left table and the child is the right table.
- Extended primary key – a combined key of every table in the data object
- Attribute – a column of any table in a data object
- Root attribute – a column of the root table
- Nested attribute – a column of one of the nested tables
- Extended table – a left-outer join of a branch leading to the child table that includes primary keys of the ancestors and the rows in the child table. For example, if the data object consists of a
    - Root table R with
        - Primary key column Pr
        - Children T1 and T2
    - T2 with
        - Primary key P2
        - Foreign key Pr
        - Join condition with R of R.Pr = T2.Pr

- Child T21

o T21 with

- Primary key P21
- Foreign key P2
- Join condition T2.P2 = T21.P2
- Columns C1, ..., Ci,

The extended T21 table would be generated as: select R.Pr, T2.P2, T21.P21, T21.C1, ..., T21.Ci from R, T2, T21 where R.Pr *= T2.Pr and T2.P2 *= T21.P2

- Extended table key – a combined key of all ancestors of a table and the table. In the previous example, the extended table key would be Pr, P2, P21

Mapping Discovery

Mapping Discovery involves the following steps:

Schema mapping

Binding Discovery

Correlation Discovery

Transform Function Discovery

Schema Mapping

Data mapping is expensive. In order to focus mapping on relevant columns, we try to identify high probability correlated attributes using schema mapping. Schema mapping relies on several techniques:

- Metadata index – a collection of words and synonyms used in table and column names and descriptions

- Type compatibility matrix
- Exception list – a collection of words that may not provide meaningful correlation

Frequently used names: names that do not present a high confidence correlation without an additional qualifier (e.g., "name", "id", "number")

Description Ignore List: words that do not provide semantic content (e.g., "the", "a", "in", "on", "for")

Metadata Index

Metadata index is the main schema-mapping tool. It is essentially a hash table of all the words encountered either as table names, column names or descriptions. Each entry in the table contains:

- A list of UDO attributes that it corresponds to and has a different relevance score for each attribute.
- A list of synonyms (pointers to their index entries) that were either pre-loaded or discovered. For each synonym, there is a synonym relevance score that reflects how closely the synonym matches a given word (e.g., "Number" and "Num" may have a relevance score of 0.9, but "Number" and "Code" of 0.6)

Entry	Attributes (Score)	Synonyms (Score)
Num	CustomerUDO.CustomerId (0.4)	Number (0.9), Id (0.5), Count (0.8), Code (0.6), Nr (0.8), Identifier (0.5)
Customer	CustomerUDO.CustomerId	Client (0.9), Cust (0.9), KUN

	(0.5), CustomerUDO.CustomerAddress (0.4)	(0.8)

Metadata Index Construction

Initially, the index is preloaded with a set of common term synonyms (e.g., "Num", "Number", "Id", "Identifier", "Code", etc.). These entries are linked together with appropriate relevance scores and do not have any UDO attributes.

Every time we perform mapping on data objects, the index gets populated as follows:

For each new attribute added to a UDO, add UDO attribute name to the index

Add column name to the index

If the name is a frequently used name from the exception list (e.g., id), assign it relevance score of 0.5

If the name is not a frequently used name, split it into words using delimiters. E.g., "Customer Name" gets split into "Customer", "Name".

Each word gets added to the index with a score of 1/#words – e.g., .5 and .5. Supported delimiters include ' ', '-', '_' and caps (e.g., CustomerName)

Break column description into words.

Remove the words in the Description Ignore List from the list.

For each remaining word, add to the Metadata index with a relevance = 0.8/#words.

Add UDO object name to the index with relevance of 0.2/#words.

Add UDO object description to the index with relevance of 0.2/#words in the description.

If we match an existing UDO attribute to an SDO attribute, we set the multiplier to 1 and update the index.

Given a multiplier, update the index as follows. Only add entries to the index whose final relevance score > 0.1).

Add SDO column name to the index with the relevance score of multiplier/#words

Add SDO column description to the index with the relevance score of 0.8*multiplier/#words

Add SDO table name to the index with the relevance score of 0.2*multiplier /#words

Add SDO table description to the index with the relevance score of 0.2*multiplier /#words

If there are any foreign keys, add them as columns to the metadata index (repeat step 2) with a multiplier of 0.8 if identified through metadata or 0.6 if discovered

If there is a user defined datatype for the column, we add it to the index with relevance of 0.5*multiplier/#words

Schema Mapping Rules

The following rules are included as part of the invention:

- Word match rule
- Type conversion rule
- String Containment Rule – checks if string1 is contained in string2

- String Overlap Rule – checks if string1 and string2 have a common substring

In addition, the users will be able to develop arbitrarily complex custom rules.

Schema Matching Process

This process describes schema matching using word match and type conversion rules:

Start with an empty Table Relevance List RT

For each source table.

Set relevance to 0.2

Apply Word Match Rule to table name – get back a list of relevant attributes with relevancy score for each – add them to RT

Apply Word Match Rule to table description – add resulting attributes to RT

For each source column start with an empty Relevant Attribute List RA

Copy RT (Table relevance list) to the Relevant Attribute List RA

Apply Word Match Rule to column name

Set Relevance to 1

Add the results to RA

If the column has a user defined datatype, apply Word Match Rule to the datatype with relevance of 0.5

Apply Word Match Rule to column description with relevance of 0.8

Each column now has a Relevant Attribute List RA

Sort list by relevance

Remove duplicate entries – if an Attribute is in the list more than once, keep the entry with highest relevance score (OPEN ISSUE: should we add the relevance scores instead?)

For each relevant attribute in RA, perform type match using type matching rule

If the type matching rule determines that the source column and the attribute in RA can be converted into each other without any loss of data or are identical, we keep the score for the attribute as is

If the types can only be converted into each other with potential loss of data, we multiply the relevance score by 0.8

If the types cannot be converted into each other, we remove the attribute from the list

Create a target oriented relevance list TRL such that every target attribute mentioned in RL, gets it own list of relevant source attributes

For each column SC that has a relevant attribute list RA

For each target attribute TC in RA

Add SC to TC's TRL with the relevance score in RA

Binding Condition Discovery

Once we have a Relevance list, we can discover the binding condition between the UDO and an SDO. To simplify the description, this section describes the discovery of the binding condition between a source table and a target table. The described algorithm can be easily extended to the tables of a UDO and SDO or can be applies to any part of SDO or UDO sub-tree by flattening that sub-tree using an outer join technique described above.

A binding condition is a Boolean expression between the columns of the source table and the column of the target table that exclusively matches a row or a set of rows in the source table with a row or a set of rows in the target table. For example, given a source table S and a target table T, binding condition S.customerid = T.clientnum will match a row that represent GE as a customer in S with a corresponding row that represents GE as a customer in T. We refer to these corresponding rows of Source and Target tables as bound.

Note that the match has to be exclusive – in other words, a row SR that represents GE in the source can only match a single row in the target TR that represent GE and TR can only match SR. In general, the binding condition does not have to be one row to one row. It can be one row to many rows, many rows to one row or many rows to many rows as long as every row in the source row set SRS is exclusively bound to the rows in the target row set TRS and every row in TRS is exclusively bound to the rows in SRS.

Multiple binding conditions are possible between the same tables (e.g., S.custname = T.clientname may be a binding condition as can S.customerid = T.clientnum). Percentage of bound rows helps identify the best binding condition. However, multiple binding conditions are possible. In fact, sometimes we get "false" binding conditions. For example, a customer id can be assigned independently as an integer in two disparate systems such that customer id 99 on one system is for customer X, while 99 on another system is for customer Y. A binding condition relating customer ids across these systems would be a *false* binding condition.

In order to identify false binding conditions, we will use correlation discovery process to identify correlations across systems. False binding conditions are unlikely to

have any correlations, while true binding conditions will have multiple correlations. Thus the higher the number of correlation and the stronger those correlations are, the stronger is the binding condition.

Binding conditions are usually based on equality as illustrated in the following process. However, they can be any Boolean expression that can be applied to source and target tables to identify rows that are uniquely bound to each other.

Equality Based Binding Condition Discovery (EBCD1) Process

Goal: Given a source table S and target table T, discover Boolean expression B that exclusively binds rows of S to rows of T.

Because of dirty data or domain mismatch (e.g., source has data for NY, while target for US), the binding may not be perfect. We use the following thresholds to help identify meaningful matches.

- Overlap Threshold - % of rows that must be matches for a source/target column pair to be considered a potential binding condition (e.g., 40%)
- Combination Threshold - % of rows where two predicates occur together to be considered a combination

Process:

1. Create source column index table SCIT of all values in S. Each row of the column index table will contain a value Value, a column number ColNum and a count of how many times the value occurred in that column - RowCount.

2. Create a target column index table TCIT of all values in T. Each row of the column index table will contain a value, a column number and a count of how many times the value occurred in that column

3. Set NumRows to the lesser of the number of rows in S and the number of rows in T

4. Create an Overlap Column Table OCT with columns: SourceCol – the column number of the source column, TargetCol – the column number of the target column, OverlapCount – the number of overlapping rows

a. For each unique row Srow in SCIT (a unique source column number, value combination)

i. For each unique row Trow in TCIT (a unique target column number, value combination)

1. If Srow.Value = Trow.Value, set MatchRowCount to the smaller of Srow.RowCount and Trow.RowCount

2. Add MatchRowCount to OverlapCount in OCT[SourceCol][TargetCol]

5. Move high probability binding condition pairs into Predicate table with columns SourceCol and TargetCol

a. For each Source column S

i. For each target column T

1. If OCT[S][T] > numRows * OverlapThreshold, add S and T to the Predicate Table, setting SourceCol to S and TargetCol to T

2. Assign a unique number to each predicate (e.g., PredicateNum)

b. Store Predicates table in a relational database

6. Construct binding conditions. At this point, we have a set of potential predicates. Some of these may be legitimate and some may be false binding conditions. We would like to identify combinations of predicates that generate more powerful binding conditions than individual predicates.

a. Build a Source Row Index Table (SRIT) with columns Value, ColNum and RowNum

i. Make a list of all source columns in Predicates Table: PredSourceCol1...PredSourceColN

ii. Select values for those columns (select PredSourceCol1, ..., PredSourceColN from S)

iii. For each row R

1. For each column C

a. Add a row to SRIT where Value = the value of C, ColNum = the column number of C and RowNum = row number of R

iv. Store SRIT in a relational database

b. Build a Target Row Index Table (TRIT) with columns Value, ColNum and RowNum

i. Make a list of all Target columns in Predicates Table: PredTargetCol1...PredTargetColN

ii. Select values for those columns (select PredTargetCol1, ..., PredTargetColN from S)

iii. For each row R

1. For each column C

a. Add a row to TRIT where Value = the value of C, ColNum = the column number of C and RowNum = row number of R

iv. Store TRIT in a relational database

c. Create a RowMatchTable (or view) that identifies which predicates occur on which rows

i. Execute the following query against a database containing SRIT, TRIT and Predicates tables: Select Predicates.PredNum, SRIT.RowNum as SourceRow, TRIT.RowNum as TargetRow from Predicates, SRIT, TRIT where Predicates.SourceCol = SRIT.ColNum and Predicates.TargetCol = TRIT.ColNum and SRIT.RowNum = SRIT.RowNum

ii. Store the resulting rows in RowMatchTable(PredicateNum, SourceRow, TargetRow)

d. Identify predicates that have high co-occurrence - for each predicate combination, count how many times predicates occur in the same row

i. Issue the following SQL query:

select RMT1.PredicateNum as Pred1, RMT2.PredicateNum as Pred2, count (*) as RowCount from RowMatchTable RMT1,

RowMatchTable RMT2 where RMT1.SourceRow = RMT2.SourceRow and RMT1.TargetRow = RMT2.TargetRow group by RMT1.PredicateNum, RMT2.PredicateNum

ii. Load the rows where RowCount > NumRows * CombinationThreshold into Predicate Combinations Table PCT

iii. Eliminate duplicates from PCT by deleting all rows where Pred1 > Pred2

e. Create binding conditions by combining predicates with high co-occurrence

i. Read rows from PCT ordered by Pred1. For each new value P1 of Pred1

1. Add P1 to PredicateList

2. For every row R with Pred1=P1, get P2 = value of Pred2

a. Add P2 to PredicateList

3. Generate every combination of Predicates in PredicateList (e.g., if PredicateList contains predicates 3, 5, and 11, generate ({3}, {5}, {11}, {3, 5}, {3, 11}, {5, 11} and {3, 5, 11})

4. Add each unique new combinations to the BindingConditionsList (i.e., do not add duplicates)

f. Generate Binding Condition strings

i. For every combination C of predicates in BindingConditionsList

1. For each predicate P in C
    a. Set SourceColName to the source column name for that predicate
    b. Set TargetColName to the target column name for that predicate
    c. Generate a predicate string of the form "<SourceColName> = <TargetColName>"
2. Add predicate to the BindingConditionString
    a. If BindingConditionString is empty, BindingConditionString = PredicateString
    b. If BindingConditionString is not empty, BindingConditionString = "<BindingConditionString> and <PredicateString>"

ii. Add BindingConditionString to the list of binding conditions for S and T

7. Validate Binding Conditions using Correlations
    a. For each Binding Condition
        i. Perform Correlation Discovery Process (CD1)
        ii. If CD1 does not return any correlations, remove Binding Condition from the Binding Condition List
        iii. Otherwise, for each Correlation C discovered by CD1
            1. Set BindingConditionWeight = BindingConditionWeight + CorrelationWeight for C

b. Order binding conditions in BindingConditionWeight order (i.e., the one with the most correlations wins)

Correlation Discovery

This process describes the discovery of correlation between Source and Target columns of bound tables. A set of source columns is said to be correlated to a set of target columns, if in every row of a source table, the value of a the source columns always corresponds to the same value of the target columns in a bound row of the target table. For example, Source.Region is correlated to Target.District if for every row of Source where Region = 'R1', in the bound row of the Target, District = 'East'. Since data may be dirty or out of sync, we do not expect the correlation to hold 100% and use a threshold to determine whether to accept a partial correlation.

Another way to think about this is if we join Source and Target on the binding condition, there will be a functional dependency between the correlated source and target columns. Note that we do not need to know how to transform the source columns into target columns, just that they correspond.

A transformation is a correlation where we actually know how to generate the target column value from the bound source column values. For example, if Target.AreaCode = substring(Source.Phone, 1, 3) (first three characters of the Source.Phone field), there is a correlation between Target.AreaCode and Source.Phone and the transformation function is substring(Source.Phone, 1, 3). While a transformation is strictly a subset of correlation, in the rest of this document, we will refer to correlations where we do not know the transformation function as correlation and the ones where we do as transformations.

Correlation Discovery (CD1) Process

Goal: Given source table S, target table T and Binding condition B, discover all correlations between columns in S and columns in T.

To help with the process, we will use the following arrays and variables:

- Int CorrelationCount[num source columns][num target columns] – keeps correlation counters for each {S.Ci, T.Cx} combination as CorrelationCount[i][x]
- Int NonUniqueRowCount [num source columns] – keeps track of number of rows that contain non-unique values for that column for each source column
- Int TotalRowCount – contains the number of bound row combinations.

We also use the following thresholds:

- UniqueColumnThreshold - % of non-unique rows that gives meaningful correlation – e.g., 60%
- CorrelationThreshold - % of correlated rows that gives a meaningful correlation – e.g., 40%

For each source column S.Ci, target column T.Cx combination, we will summarize the maximum count of a distinct value of how many times a value of T.Cx occurs for the each value of S.Ci. For example, if S.Ci = 5 occurs in 5 rows. In 3 of these rows T.Cx = 'abc' and in 2 rows – 'xyz'; the max count of any value is 3 (for value 'abc'), so we will add 3 to the count of correlated rows between

S.Ci and T.Cx.

For each source column S.Ci, issue the following SQL query:

Select S.Ci, T.C1, ..., T.Cz from S, T where B order by Ci

The query will return a stream of values ordered by a value of S.Ci.

a. For each row, increment TotalRowCount

b. For each unique value of S.Ci

i. If there is only 1 row for S.Ci – skip – go to the next value of S.Ci

ii. If there are multiple rows

1. Increment NonUniqueRowCount[i] by number of rows

2. For each target column T.Cx

a. For each unique value U in T.Cx,

i. Set RowCount = count of how many times it occurs for the same value of S.Ci

ii. Add RowCount to a list of row counts RowCountsList

b. Set MaxCount = Select the largest row count from RowCountsList

c. Add MaxCount to CorrelationCount[i][x]

2. For each source column S.Ci

a. Eliminate unique source columns

i. If NonUniqueRowCount[i] < TotalRowCount * UniqueColumnThreshold, skip this column

b. For each target column T.Cx

i. If CorrelationCount[I][X] > TotalRowCount * CorrelationThreshold, create a correlation between S.Ci and T.Cx with CorrelationWeight of TotalRowCount / CorrelationCount[I][X]

Transformation Discovery

Goal: Given a binding condition (BINDING_CONDITION) and correlation between one or more source columns and a target column, discover the transformation function to generate the target column value from the source column values.

Different approaches can be applied for different types of columns:

1. Numeric columns
   a. Determine ratio
   b. Determine difference
   c. Convert to strings and apply string processes
2. Date, Time and Datetime columns
   a. Determine interval (difference)
3. String columns
   a. Apply PTD1 process to discover Positional Transformations
   b. Apply TTD1 process to discover Token Transformations
   c. If numeric, convert to a number and apply numeric tests

d. If a date, time or datetime, convert to date and apply date processes

Positional Transformation Discovery Process 1 (PTD1)

To execute the process, we will use the following arrays, variables and structures:

- Int CharMap[128][MAX_COL_LEN] – for each character, this array has a list of locations in the target column value where that character occurs
- Int CharMapCount[128] – for each character, this array contains a count of locations where that char occurs (i.e., the length of location list in CharMap)
- Int CharConstant [MAX_COL_LEN] [128] – for each location, count how many times each character occurs in each location
- Char Constant [MAX_COL_LEN] – the constant character for each location, '\0' otherwise
- Int ForwardCharMatch[MAX_COL_LEN][MAX_NUM_COLUMNS][MAX_COL_LEN] – for each location in target column value, keeps a count of how many times there is a matching character for each location in the source column value
- Int ReverseCharMatch[MAX_COL_LEN][MAX_NUM_COLUMNS][MAX_COL_LEN] – for each location in target column value, keeps a count of how many

times there is a matching character for each reverse location in the source column value. Reverse location is counted from the end – e.g., 0 means the last character, 1 means the one before last, etc.

- Int matchThreshold - % of rows that need to have a match for a location for it to be considered a match. Assumed to be > 50%

- Int constantThreshold - % of rows that need to have the same character in a target location for it to be considered a constant. Assumed to be > 50%

- Int variableThreshold - % of rows that need to have a fixed value of target column for it to be considered variable length. VariableThreshold should be < ConstantThreshold

- Int NumberOfRows – number of rows in the target that have a matching row in source tables given the binding condition

- Structure Match – consists of the following fields
  - Type – one of Forward, Reverse, Constant
  - int column – source column
  - int location – source location
  - char constant

- Match Matches[MAX_COL_LEN][MAX_COL_LEN] – array of matches that has a list of matches for each target location
- MatchCount[MAX_COL_LEN] – number of matches for each target location in Matches array

Process

1. For each target column, read all correlated source columns in a single select (i.e., select sourcecolumn1, sourcecolumn2, … , targetcolumn from sourcetable1, sourcetable2, …, targettable where BINDING_CONDITION order by targetcolumn). For each row
    a. Increment NumberOfRows
    b. Get the value of the target column – TVALUE
    c. SetCharConstant(TVALUE)
    d. If TVALUE is not the same as previous TVALUE (PTVALUE)
        i. Clear CharMap, CharMapCount
        ii. SetCharMap(TVALUE)
        iii. Reset PTVALUE = TVALUE
    e. For each source column
        i. Get value of source column UNTRIMMED_SVALUE
        ii. Trim blanks from the end SVALUE = rtrim(UNTRIMMED_SVALUE, ' ')
        iii. Get the column number SCOL
        iv. For each character in SVALUE
            1. Get char value: ScharValue
            2. Get char location: SForwardCharLocation

3. Get reverse char location: SreverseCharLocation = Length(SVALUE) - SForwardCharLocation

4. Get count of entries in CharMap – TlocationCount = CharMapCount[ScharValue]

5. For each entry Entry in CharMap

a. Get target location TcharLocation = CharMap[Entry]

b. Increment forward match count ForwardCharMatch[TcharLocation][SCOL][SForward CharLocation] ++

c. Increment reverse match count ReverseCharMatch[TcharLocation][SCOL][SReverse CharLocation] ++

2. Identify constants

a. For each location ConstantLocation in CharConstants

i. Set Constant[ConstantLocation] = '\0' – to indicate there is no constant

ii. Set LocationCount = 0

iii. For each character ConstantCharacter

1. Get the count ConstantCount of times character ConstantCharacter was in location ConstantLocation in target values

2. Set LocationCount = LocationCount + ConstantCount

3. If ConstantCount > ConstantThreshold*NumberOfRows
    a. Create a Match record M
        i. M.Type = Constant
        ii. M.Constant = ConstantCharacter
    b. Add M to Matches
        i. count = MatchCount[ConstantLocation]
        ii. Matches[ConstantLocation][count] = M
        iii. MatchCount[ConstantLocation] = count + 1
    iv. If LocationCount < ConstantThreshold – we have reached the point where we will not find any more constants because we do not have enough values to overcome the threshold
        1. If LocationCount > VariableConstantThreshold – that means we have a variable length column and cannot do positional matching. Stop PTD1
        2. Else if LocationCount < VariableConstantThreshold, the values that are longer then LocationCount are too few to worry about. We should fix TargetColumnLength = ConstantLocation
        3. Stop constant identification
3. For each target location TLOCATION, find the best match

a. For each source column SCOL

i. For each source location SLOCATION

1. get ForwardMatchCount = ForwardCharMatch[TLOCATION][SCOL][SLOCATION]

2. if ForwardMatchCount > matchThreshold

a. Create a Match record M

i. M.Type = Forward

ii. M.Col = SCOL

iii. M.Location = SLOCATION

b. Add M to Matches

i. count = MatchCount[TLOCATION]

ii. Matches[TLOCATION][count] = M

iii. MatchCount[TLOCATION] = count + 1

3. get ReverseMatchCount = ReverseCharMatch[TLOCATION][SCOL][SLOCATION]

4. if ReverseMatchCount > matchThreshold

a. Create a Match record M

i. M.Type = Reverse

ii. M.Col = SCOL

iii. M.Location = SLOCATION

b. Add M to Matches

i. count = MatchCount[TLOCATION]

ii. Matches[TLOCATION][count] = M

iii. MatchCount[TLOCATION] = count + 1

4. Generate functions

a. If MatchCount[TLOCATION] is 0 for any TLOCATION = 0.. TargetColumnLength

i. STOP – we could not discover any positional transformations

b. For each combination COMBINATION of matches (e.g., if TargetColumnLength = 2 and MatchCount[0] = 2 and MatchCount[1] = 3, we will have 6 permutations: {Match[0][0], Match[1][0]}, {Match[0][0], Match[1][1]} ,...}. COMBINATION is array of Match records

i. Start with FUNCTION = Empty string

ii. Convert consecutive forward char matches into substring functions

1. if there are 1 or more Matches in locations Li=L1..LN such that COMBINATION[Li].Type = Forward and COMBINATION[Li].Col = COMBINATION[Li+1].Col and COMBINATION[Li].Location = COMBINATION[Li+1].Location – 1

a. Create a substring function SUBSTRING = substring(ColName[Col], L1, LN-L1+1)

b. Add substring to function: FUNCTION = FUNCTION + SUBSTRING

iii. Convert consecutive constants into constant strings